



Programación Python para Big Data

Lección 6: VAEX y DASK

ÍNDICE

Lección 6. – VAEX y DASK.....	2
Presentación y objetivos.....	2
1. Herramientas para enormes DataSets	3
2. Tiempos de lectura y procesamiento: Comparativa.....	4
3. Introducción a VAEX.....	5
4. Introducción a DASK.....	7
5. Dataset útil para el aprendizaje de BIG DATA.....	8
6. Abrimos ese archivo con Microsoft Excel	10
7. Explicación del “Experimento” a realizar	11
8. El Ordenador influye en el “Experimento”	12
9. Abrimos ese DataSet con Pandas en Jupyter	13
10. Abrimos ese DataSet con DASK (leyendo .CSV)	17
11. Abrimos ese DataSet con VAEX (leyendo .CSV)	20
12. HDF5 y APACHE PARQUET: Breve Introducción.....	22
13. Abrimos ese DataSet con VAEX (leyendo .HDF5)	23
14. VAEX: Otra forma de Resolverlo.....	26
15. Puntos clave.....	29

Lección 6. – VAEX y DASK

PRESENTACIÓN Y OBJETIVOS

En la presente lección de esta asignatura veremos 2 posibles alternativas a PANDAS para trabajar con sets de datos de gran tamaño.

Estas 2 herramientas son: VAEX y DASK. Existen más, algunas de ellas serán comentadas, pero las que presentamos son muy buenas alternativas.



Objetivos

- | Conocer VAEX y DASK como alternativas a PANDAS cuando el set de datos es muy grande.
- | Comprender una posible forma de analizar las herramientas BIG DATA en base al tiempo de ejecución de las celdas.

1. HERRAMIENTAS PARA ENORMES DATASETS

PANDAS es una de las maravillosas herramientas que disponemos en DATA SCIENCE. Millones de Data Scientists lo usan a diario. Es una de las herramientas que merece la pena conocer muy muy bien.

En aquellos casos que necesitamos trabajar con sets de datos de varios millones de filas es necesario hacerlo de forma rápida, y para ello han salido algunas herramientas muy útiles.

Algunas de las posibles herramientas que tenemos son:

Vaex

Lo veremos en este tema.

<https://vaex.io/docs/datasets.html>

Dask

Lo veremos en este tema

<https://dask.org/>

Rapids

No lo veremos en este manual

<https://rapids.ai/>

Modin

No lo veremos en este manual

<https://modin.readthedocs.io/en/latest/>

Ray

No lo veremos en este manual

<https://ray.io/>

Koalas

No lo veremos en este manual

<https://koalas.readthedocs.io/en/latest/>

2. TIEMPOS DE LECTURA Y PROCESAMIENTO: COMPARATIVA

Una de las formas de analizar qué herramienta queremos es medir el tiempo que tarda cada herramienta en procesar nuestros datos.

Para ello haremos una comparativa de tiempos de lectura de un archivo .CSV entre otras cosas.

Podríamos hacer una comparativa teniendo en cuenta más cosas, pero quizá de esta forma ya se puede entender un poco la necesidad de emplear otras herramientas, y no solamente PANDAS a medida que tenemos un volumen de datos mayor del habitual.

De entre todas las opciones nos hemos decantado por VAEX y por DASK para hacer este estudio. Pensamos que son 2 muy buenas alternativas a PANDAS.

Trataremos de hacer comparativas teniendo en cuenta incluso el tipo de Disco Duro del ordenador.

Existen varias formas de hacer un análisis de tal forma que trataremos de hacerlo de un modo holístico.

Pero, antes de nada.. ¿Qué es VAEX? ¿Qué es DASK?

¡Comencemos paso a paso!

3. INTRODUCCIÓN A VAEX

Características:

- | VAEX es una librería similar a PANDAS, que es ideal para grandes datasets.
- | Dispone de integración con Jupyter también.
- | Instalación, como siempre:
<https://pypi.org/project/vaex/>
- | **Es una librería que permite trabajar con datos tan grandes como el tamaño del Disco Duro que se tenga, y en una sola máquina.**
- | **Sostiene que el Escalado Horizontal (agregar) más ordenadores/nodos NO es necesario para la mayoría de tareas en ciencia de datos.**
- | Es una implementación de DataFrame distinta de la que existía hasta entonces, creada desde cero, que permite trabajar con DataFrames de cientos de millones o miles de millones de filas.
- | Emplea algoritmos de C++ sobre la API establecida de Pandas para mayor eficiencia.
- | **Podría ser capaz de procesar 1000 Millones de filas en 1-2 segundos**
- | Admite “evaluaciones perezosas” que, aunque no usa el método `.compute()` es algo no igual, pero también de modo que las operaciones se retrasan pero que de algún modo agilizan.

(No tiene especial importancia saber qué es eso en este momento).

- | Los datos para VAEX deberían estar en un formato mapeable en memoria en el disco.
- | Hay que tener en cuenta la velocidad de lectura en los diferentes tipos de Disco Duro: HDD Vs SSD. Por ello es conveniente tener en cuenta la diferencia en Velocidad al trabajar con SSD frente a HDD.
- | Por todo ello podemos hacer todo el proceso en una sola máquina incluso con nuestro propio ordenador de un modo rápido.
- | El formato de datos preferido de VAEX es “HDF5” o incluso Apache Arrow, aunque también puede leer Apache Parquet
- | VAEX puede leer .CSV
- | VAEX puede leer .JSON
- | VAEX espera que todos los datos estén en una sola máquina.
- | Es posible que con VAEX exista alguna cosa que no existe implementación, por ser tan reciente, pero probablemente vaya en aumento esta tecnología.

4. INTRODUCCIÓN A DASK

Características:

- | Realmente se refiere a `Dask.DataFrame` que es una librería construida sobre PANDAS y DASK
<https://docs.dask.org/en/latest/dataframe.html>
- | **Un `Dask.DataFrame` es separado en muchos pequeños Pandas DataFrames, Lo que ocurre es que cada pequeño DataFrame hace un cálculo y después se une el resultado final.**
(Paraleliza cálculos de esta forma)
- | Varias librerías como SK-Learn, o algoritmos como XGBoost pueden emplearlo
- | Admite “evaluaciones perezosas” en el caso de DASK hasta que el usuario emplea el método “.compute()” el cálculo no se realiza.
Es algo que vamos a ver en el propio ejemplo.
- | El formato de datos preferido de DASK es “Apache Parquet” aunque no lo veremos en este manual de forma práctica, solo brevemente en teoría.
- | DASK puede leer .CSV
- | DASK puede leer .JSON

5. DATASET ÚTIL PARA EL APRENDIZAJE DE BIG DATA

En el siguiente Link podemos encontrar varios sets de datos de gran volumen y que se pueden considerar como un set de datos de referencia a la hora de impartir formación en Big Data.

<https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page>

Tenemos en la misma web



Figura 5.1: DataSet típico en aprendizaje de Big Data (parte 1)

La web nos da opción de traducción, pero trataremos de no hacerlo.

En la misma web, si hacemos "scroll down", si nos movemos hacia abajo en esa misma página, tenemos lo siguiente:

January <ul style="list-style-type: none"> • Yellow Taxi Trip Records (CSV) • Green Taxi Trip Records (CSV) • For-Hire Vehicle Trip Records (CSV) • High Volume For-Hire Vehicle Trip Records (CSV) February <ul style="list-style-type: none"> • Yellow Taxi Trip Records (CSV) • Green Taxi Trip Records (CSV) • For-Hire Vehicle Trip Records (CSV) • High Volume For-Hire Vehicle Trip Records (CSV) March <ul style="list-style-type: none"> • Yellow Taxi Trip Records (CSV) • Green Taxi Trip Records (CSV) • For-Hire Vehicle Trip Records (CSV) • High Volume For-Hire Vehicle Trip Records (CSV) April <ul style="list-style-type: none"> • Yellow Taxi Trip Records (CSV) • Green Taxi Trip Records (CSV) • For-Hire Vehicle Trip Records (CSV) • High Volume For-Hire Vehicle Trip Records (CSV) 	July <ul style="list-style-type: none"> • Yellow Taxi Trip Records (CSV) • Green Taxi Trip Records (CSV) • FHV Trip Records (CSV) • High Volume For-Hire Vehicle Trip Records (CSV) August <ul style="list-style-type: none"> • Yellow Taxi Trip Records (CSV) • Green Taxi Trip Records (CSV) • FHV Trip Records (CSV) • High Volume For-Hire Vehicle Trip Records (CSV) September <ul style="list-style-type: none"> • Yellow Taxi Trip Records (CSV) • Green Taxi Trip Records (CSV) • FHV Trip Records (CSV) • High Volume For-Hire Vehicle Trip Records (CSV) October <ul style="list-style-type: none"> • Yellow Taxi Trip Records (CSV) • Green Taxi Trip Records (CSV) • FHV Trip Records (CSV) • High Volume For-Hire Vehicle Trip Records (CSV)
--	---

Figura 5.2: DataSet típico en aprendizaje de Big Data (parte 2)

En nuestro caso vamos a trabajar con uno de ellos concreto que comentaremos, pero se refiere a datos de 2018, por ser algo mayor en tamaño que los más recientes.

6. ABRIMOS ESE ARCHIVO CON MICROSOFT EXCEL

Con Excel no disponemos de tantas filas en la misma pestaña, de tal manera que no podremos (a priori) procesar este volumen de datos con Excel, por lo menos no es la mejor opción tal y como veremos.

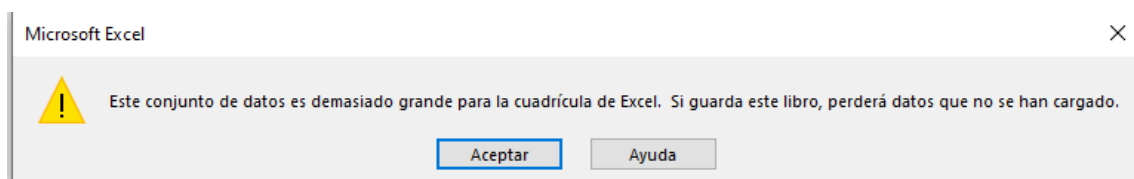


Figura 6.1: Excel y DataSets grandes (parte 1)

	A	B	C	D	E	F	G	H
1048553	2,2020-01-07 07:42:27,2020-01-07 08:10:30,6,3.92,1,N,170,238,1,19.5,0,0.5,4.56,0,0.3,27.36,2.5							
1048554	1,2020-01-07 07:34:04,2020-01-07 07:46:40,1,2.80,1,N,141,74,1,11.5,2.5,0.5,2.95,0,0.3,17.75,2.5							
1048555	1,2020-01-07 07:50:09,2020-01-07 07:53:08,3,.40,1,N,41,42,2,4,0,0.5,0,0.3,4.8,0							
1048556	2,2020-01-07 07:37:48,2020-01-07 07:41:51,1,.96,1,N,140,229,1,5.5,0,0.5,1.32,0,0.3,10.12,2.5							
1048557	2,2020-01-07 07:48:50,2020-01-07 07:56:44,1,1.32,1,N,229,170,1,7,0,0.5,2.06,0,0.3,12.36,2.5							
1048558	2,2020-01-07 07:11:46,2020-01-07 07:26:07,1,3.06,1,N,24,230,1,13,0,0.5,1.63,0,0.3,17.93,2.5							
1048559	2,2020-01-07 07:35:21,2020-01-07 07:41:39,1,1.10,1,N,142,161,1,6,0,0.5,0.93,0,0.3,10.23,2.5							
1048560	2,2020-01-07 07:42:36,2020-01-07 07:45:46,1,.39,1,N,161,237,1,4,0,0.5,1.46,0,0.3,8.76,2.5							
1048561	2,2020-01-07 07:50:21,2020-01-07 07:56:29,1,.70,1,N,237,140,1,6,0,0.5,1,0,0.3,10.3,2.5							
1048562	2,2020-01-07 07:08:59,2020-01-07 07:20:41,1,4.48,1,N,152,50,1,15,0,0.5,0,0.3,18.3,2.5							
1048563	2,2020-01-07 07:23:17,2020-01-07 07:27:28,1,1.54,1,N,50,246,1,6,0,0.5,1.86,0,0.3,11.16,2.5							
1048564	2,2020-01-07 07:43:05,2020-01-07 07:54:41,1,1.22,1,N,164,137,1,8.5,0,0.5,3.54,0,0.3,15.34,2.5							
1048565	1,2020-01-07 07:57:09,2020-01-07 08:03:17,1,.60,1,N,25,33,1,6,0,0.5,1.35,0,0.3,8.15,0							
1048566	1,2020-01-07 07:08:43,2020-01-07 07:15:49,1,1.00,1,N,100,161,1,6,2.5,0.5,1.85,0,0.3,11.15,2.5							
1048567	1,2020-01-07 07:43:43,2020-01-07 07:46:23,1,.70,1,N,79,137,2,4.5,2.5,0.5,0,0.3,7.8,2.5							
1048568	1,2020-01-07 07:49:10,2020-01-07 07:59:31,1,5.10,1,N,233,261,2,16,2.5,0.5,0,0.3,19.3,2.5							
1048569	2,2020-01-07 07:29:16,2020-01-07 07:44:33,1,1.86,1,N,211,234,2,11,0,0.5,0,0.3,14.3,2.5							
1048570	2,2020-01-07 07:46:11,2020-01-07 07:55:45,1,1.23,1,N,234,233,1,7.5,0,0.5,2.16,0,0.3,12.96,2.5							
1048571	1,2020-01-07 07:26:52,2020-01-07 07:31:50,2,1.90,1,N,262,229,1,7.5,2.5,0.5,2.15,0,0.3,12.95,2.5							
1048572	1,2020-01-07 07:33:47,2020-01-07 07:38:56,1,.70,1,N,229,140,2,5.5,2.5,0.5,0,0.3,8.8,2.5							
1048573	1,2020-01-07 07:40:48,2020-01-07 07:50:38,1,1.30,1,N,140,162,1,8.5,2.5,0.5,2.5,0,0.3,14.3,2.5							
1048574	1,2020-01-07 07:19:09,2020-01-07 07:28:24,1,1.40,1,N,48,161,1,7.5,2.5,0.5,2.16,0,0.3,12.96,2.5							
1048575	1,2020-01-07 07:41:00,2020-01-07 07:45:59,3,.70,1,N,233,162,1,5.5,2.5,0.5,1.76,0,0.3,10.56,2.5							
1048576	1,2020-01-07 07:39:48,2020-01-07 07:43:45,1,1.50,1,N,79,137,1,6,2.5,0.5,1.86,0,0.3,11.16,2.5							

Figura 6.2: Excel y DataSets grandes (parte 2)

7. EXPLICACIÓN DEL “EXPERIMENTO” A REALIZAR

Para nuestro caso particular vamos a resolver el presente ejercicio con Jupyter Notebook, Entorno de Desarrollo ya empleado anteriormente.

Lo que haremos será tratar de medir el tiempo de ejecución de cada celda.

La parte principal en nuestro tema será tratar de ver el tiempo de lectura del archivo .CSV, aunque también será relevante analizar los tiempos de ejecución de celdas donde hay estadística y cálculos más complejos.

Es importante resaltar lo siguiente, pudiéramos usar otros Entornos de Desarrollo para ello, y usar “time”, lo escribiríamos antes y después del código que queremos ejecutar y restaríamos uno del otro para saber el tiempo de ejecución.

En Jupyter podemos saber el tiempo de ejecución de cada celda de un modo mucho más simple, es decir, con “%%time” en la primera línea de cada celda.



Presta atención

Es importante destacar que esa instrucción no debe estar en un punto intermedio de la celda, para que pueda ser reconocida correctamente.

Haremos unos ejemplos con PANDAS, VAEX y DASK.

Posteriormente en la actividad podrás practicar con alguna herramienta más.

8. EL ORDENADOR INFLUYE EN EL “EXPERIMENTO”

Tal y como se ha mencionado previamente, la potencia del ordenador va a influir en la velocidad de ejecución de las celdas.

A continuación, mostramos un pantallazo hecho del propio Jupyter donde hemos reflejado una comparativa de los equipos empleados.

Lo ideal habría sido extraer el Disco Duro HDD, por un SSD, con el “clonado” correspondiente, pero por varios motivos ha sido realizado con el mismo Ordenador, pero usando un Disco Duro Externo SSD. (Configurando un arranque con el SSD desde la BIOS. (No tiene importancia para este tema el modo en que ha sido llevado a cabo).

No obstante, la diferencia de tiempos es muy llamativa (aunque sí, LINUX acompañaba como Sistema Operativo al Disco Duro SSD).

¡No obstante, VAEX es sinceramente impresionante !

Ordenador con el que se ha hecho el experimento

Características:

- AMD (up to 2.4 GHz)
- 16 GB DDR3
- 1000 GB HDD con 600 GB libres
- Sistema Operativo: Windows
- Empleamos el archivo mencionado en el manual. (DATOS DEL 2018)
- En mi caso: C:\Users\Manut\Desktop\apuntes_big_data_2\TEMA_6\data

Ya podemos intuir que no siendo SSD no se obtengan buenos tiempos.

NOTA IMPORTANTE:

- **El mismo experimento ha sido hecho con un disco duro externo**
- **Características: SSD de 1 TB con sistema operativo LINUX.**
- **Los tiempos de cada celda han sido descritos**

Figura 8.1: Ordenador empleado en la prueba

9. ABRIMOS ESE DATASET CON PANDAS EN JUPYTER

Pandas

```
In [1]: %%time
import pandas as pd
```

Wall time: 1.95 s

Tiempos:

- Con 1TB HDD sobre Windows: 1.95 segundos (alguna vez hasta 12 segundos)
- Con 1TB SSD sobre Linux: 638 milisegundos

Figura 9.1: PANDAS para leer el DataSet (parte 1)

```
In [2]: %%time
df_pandas = pd.read_csv("C:/Users/Manut/Desktop/apuntes_big_data_2/TEMA_6/data/yellow_tripdata_2018-01.csv")
df_pandas.head()
```

Wall time: 1min 6s

Out[2]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocation
0	1	2018-01-01 00:21:05	2018-01-01 00:24:23	1	0.5	1	N	
1	1	2018-01-01 00:44:55	2018-01-01 01:03:05	1	2.7	1	N	2
2	1	2018-01-01 00:08:26	2018-01-01 00:14:21	2	0.8	1	N	2
3	1	2018-01-01 00:20:22	2018-01-01 00:52:51	1	10.2	1	N	1
4	1	2018-01-01 00:09:18	2018-01-01 00:27:06	2	2.5	1	N	2

Tiempos:

- Con 1TB HDD sobre Windows: 1 minuto 6 segundos (a veces algo más)
- Con 1TB SSD sobre Linux: 56.6 segundos

Figura 9.2: PANDAS para leer el DataSet (parte 2)

```
In [3]: %%time
df_pandas.describe()
```

Wall time: 11.7 s

Out[3]:

	VendorID	passenger_count	trip_distance	RatecodeID	PULocationID	DOLocationID	payment_type
count	8.759874e+06	8.759874e+06	8.759874e+06	8.759874e+06	8.759874e+06	8.759874e+06	8.759874e+06
mean	1.560940e+00	1.606855e+00	2.804001e+00	1.039545e+00	1.644585e+02	1.627269e+02	1.310611e+00
std	4.962724e-01	1.258464e+00	6.412346e+01	4.450700e-01	6.636021e+01	7.031164e+01	4.817818e-01
min	1.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
25%	1.000000e+00	1.000000e+00	9.100000e-01	1.000000e+00	1.160000e+02	1.130000e+02	1.000000e+00
50%	2.000000e+00	1.000000e+00	1.550000e+00	1.000000e+00	1.620000e+02	1.620000e+02	1.000000e+00
75%	2.000000e+00	2.000000e+00	2.840000e+00	1.000000e+00	2.340000e+02	2.340000e+02	2.000000e+00
max	2.000000e+00	9.000000e+00	1.894838e+05	9.900000e+01	2.650000e+02	2.650000e+02	4.000000e+00

Tiempos:

- Con 1TB HDD sobre Windows: 11.7 segundos
- Con 1TB SSD sobre Linux: 7.76 segundos

Figura 9.3: PANDAS para leer el DataSet (parte 3)

```
In [4]: %%time
df_pandas.fare_amount.value_counts()

Wall time: 555 ms

Out[4]: 6.00      473234
        5.50      465176
        6.50      461927
        7.00      446376
        5.00      433249
        ...
        30.60         1
        2409.00        1
        168.88         1
        201.50         1
        33.96          1
Name: fare_amount, Length: 1714, dtype: int64
```

Tiempos:

- Con 1TB HDD sobre Windows: 555 milisegundos
- Con 1TB SSD sobre Linux: 313 milisegundos

Figura 9.4: PANDAS para leer el DataSet (parte 4)

```
In [5]: %%time
# casi 9 millones de filas en este caso
len(df_pandas), df_pandas.shape
```

Wall time: 0 ns

```
Out[5]: (8759874, (8759874, 17))
```

Tiempos:

- Con 1TB HDD sobre Windows: 0 nanosegundos
- Con 1TB SSD sobre Linux: 46.7 microsegundos

Figura 9.5: PANDAS para leer el DataSet (parte 5)


```
In [6]: %%time
df_pandas.tail()
```

Wall time: 0 ns

Out[6]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID
8759869	1	2018-01-31 23:21:35	2018-01-31 23:34:20	2	2.80	1
8759870	1	2018-01-31 23:35:51	2018-01-31 23:38:57	1	0.60	1
8759871	2	2018-01-31 23:28:00	2018-01-31 23:37:09	1	2.95	1
8759872	2	2018-01-31 23:24:40	2018-01-31 23:25:28	1	0.00	1
8759873	2	2018-01-31 23:28:16	2018-01-31 23:28:38	1	0.00	1

Tiempos:

- Con 1TB HDD sobre Windows: 0 nanosegundos
- Con 1TB SSD sobre Linux: 215 microsegundos

Figura 9.6: PANDAS para leer el DataSet (parte 6)

Para el caso de PANDAS hemos visto que es más rápido también si empleamos un ordenador con Disco Duro SSD y LINUX como Sistema Operativo, no obstante, no tiene tanta relevancia.

Podemos usar WINDOWS y HDD para trabajar con PANDAS, en muchos casos, y cuando necesitemos algo escalable, quizá debemos tener en cuenta la utilización de otra herramienta.

10. ABRIMOS ESE DATASET CON DASK (LEYENDO .CSV)

DASK

```
In [1]: %%time
import dask.dataframe as dd
```

Wall time: 4.13 s

Tiempos:

- Con 1TB HDD sobre Windows: 4.13 segundos (promedio 6 segundos, a veces 8 segundos)
- Con 1TB SSD sobre Linux: 846 milisegundos

Figura 10.1: DASK para leer el DataSet (parte 1)

¡Ya comenzamos bien! DASK es mucho más rápido que PANDAS !

1 minuto para PANDAS, tan solo 4 segundos para DASK.

```
In [2]: %%time
df_dask = dd.read_csv("C:/Users/Manut/Desktop/apuntes_big_data_2/TEMA_6/data/yellow_tripdata_2018-01.csv",
                    assume_missing=True)
df_dask.head()
```

Wall time: 5.9 s

Out[2]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID
0	1.0	2018-01-01 00:21:05	2018-01-01 00:24:23	1.0	0.5	1.0	N	41.0
1	1.0	2018-01-01 00:44:55	2018-01-01 01:03:05	1.0	2.7	1.0	N	239.0
2	1.0	2018-01-01 00:08:26	2018-01-01 00:14:21	2.0	0.8	1.0	N	262.0
3	1.0	2018-01-01 00:20:22	2018-01-01 00:52:51	1.0	10.2	1.0	N	140.0
4	1.0	2018-01-01 00:09:18	2018-01-01 00:27:06	2.0	2.5	1.0	N	246.0

Tiempos:

- Con 1TB HDD sobre Windows: 5.9 segundos
- Con 1TB SSD sobre Linux: 1.94 segundos

Figura 10.2: DASK para leer el DataSet (parte 2)

```
In [3]: %%time
df_dask.describe()
```

Wall time: 465 ms

Out[3]: Dask DataFrame Structure:

	VendorID	passenger_count	trip_distance	RatecodeID	PULocationID	DOLocationID	payment_type	fare_amount
npartitions=1								
	float64	float64	float64	float64	float64	float64	float64	float64

Dask Name: describe-numeric, 691 tasks

Tiempos:

- Con 1TB HDD sobre Windows: 465 milisegundos
- Con 1TB SSD sobre Linux: 298 milisegundos

Figura 10.3: DASK para leer el DataSet (parte 3)

Sin hacer el .compute() no ejecutará..

```
In [4]: %%time
df_dask.describe().compute()
```

Wall time: 50 s

Out[4]:

	VendorID	passenger_count	trip_distance	RatecodeID	PULocationID	DOLocationID	payment_type	fare_amount
count	8.759874e+06	8.759874e+06	8.759874e+06	8.759874e+06	8.759874e+06	8.759874e+06	8.759874e+06	8.759874e+06
mean	1.560940e+00	1.606855e+00	2.804001e+00	1.039545e+00	1.644585e+02	1.627269e+02	1.310611e+00	1.224434e+01
std	4.962724e-01	1.258464e+00	6.412346e+01	4.450700e-01	6.636021e+01	7.031164e+01	4.817818e-01	1.168321e+01
min	1.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	-4.500000e+02
25%	1.000000e+00	1.000000e+00	1.100000e+00	1.000000e+00	1.320000e+02	1.140000e+02	1.000000e+00	6.500000e+00
50%	2.000000e+00	1.000000e+00	1.930000e+00	1.000000e+00	1.620000e+02	1.620000e+02	1.000000e+00	9.500000e+00
75%	2.000000e+00	2.000000e+00	3.600000e+00	1.000000e+00	2.340000e+02	2.340000e+02	2.000000e+00	1.450000e+01
max	2.000000e+00	9.000000e+00	1.894838e+05	9.900000e+01	2.650000e+02	2.650000e+02	4.000000e+00	8.016000e+03

Tiempos:

- Con 1TB HDD sobre Windows: 50 segundos (57.5 segundos otra de las veces)
- Con 1TB SSD sobre Linux: 36 segundos

Figura 10.4: DASK para leer el DataSet (parte 4)

Ahora sí lo hemos ejecutado.

```
In [5]: %%time
df_dask.fare_amount.value_counts()

Wall time: 5 ms

Out[5]: Dask Series Structure:
npartitions=1
int64
...
Name: fare_amount, dtype: int64
Dask Name: value-counts-aggr, 42 tasks
```

Tiempos:

- Con 1TB HDD sobre Windows: 5 milisegundos
- Con 1TB SSD sobre Linux: 3.25 milisegundos

Figura 10.5: DASK para leer el DataSet (parte 5)

Exactamente lo mismo para este caso, necesitamos `.compute()` ..

```
In [6]: %%time
df_dask.fare_amount.value_counts().compute()

Wall time: 13.1 s

Out[6]: 6.00      473234
5.50      465176
6.50      461927
7.00      446376
5.00      433249
...
60.06         1
60.30         1
60.53         1
60.55         1
8016.00        1
Name: fare_amount, Length: 1714, dtype: int64
```

Tiempos:

- Con 1TB HDD sobre Windows: 13.1 segundos
- Con 1TB SSD sobre Linux: 5.45 segundos

Figura 10.6: DASK para leer el DataSet (parte 6)

11. ABRIMOS ESE DATASET CON VAEX (LEYENDO .CSV)

VAEX desde un CSV (1ª forma)

```
In [1]: %%time
# https://pypi.org/project/vaex/
# pip install vaex
import vaex
```

Wall time: 7.64 s

Tiempos:

- Con 1TB HDD sobre Windows: 7.64 segundos (en otro intento fueron 14.7 segundos)
- Con 1TB SSD sobre Linux: 5.14 segundos

Figura 11.1: VAEX para leer el DataSet (parte 1)

```
In [2]: %%time
# Necesito añadir convert=True para que me convierta .csv en .HDF5
df_vaex = vaex.from_csv("C:/Users/Manut/Desktop/apuntes_big_data_2/TEMA_6/data/yellow_tripdata_2018-01.csv",
                        convert=True)
```

df_vaex

INFO:MainThread:numexpr.utils:NumExpr defaulting to 4 threads.

Wall time: 3min 24s

```
Out[2]:
```

	#	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PUL
	0	1	'2018-01-01 00:21:05'	'2018-01-01 00:24:23'	1	0.5	1	'N'	
	1	1	'2018-01-01 00:44:55'	'2018-01-01 01:03:05'	1	2.7	1	'N'	
	2	1	'2018-01-01 00:08:26'	'2018-01-01 00:14:21'	2	0.8	1	'N'	
	3	1	'2018-01-01 00:20:22'	'2018-01-01 00:52:51'	1	10.2	1	'N'	
	4	1	'2018-01-01 00:09:18'	'2018-01-01 00:27:06'	2	2.5	1	'N'	

	8,759,869	1	'2018-01-31 23:21:35'	'2018-01-31 23:34:20'	2	2.8	1	'N'	
	8,759,870	1	'2018-01-31 23:35:51'	'2018-01-31 23:38:57'	1	0.6	1	'N'	
	8,759,871	2	'2018-01-31 23:28:00'	'2018-01-31 23:37:09'	1	2.95	1	'N'	
	8,759,872	2	'2018-01-31 23:24:40'	'2018-01-31 23:25:28'	1	0.0	1	'N'	
	8,759,873	2	'2018-01-31 23:28:16'	'2018-01-31 23:28:38'	1	0.0	1	'N'	

Tiempos:

- Con 1TB HDD sobre Windows: 3 minutos 24 segundos (alguna vez mucho menos)
- Con 1TB SSD sobre Linux: 1 minuto 10 segundos

Figura 11.2: VAEX para leer el DataSet (parte 2)

Cuando no indicamos “convert = True” hace lo mismo pero no transforma en .HDF5 y en ese caso tardaba alrededor de 1 minuto para WINDOWS.

En este caso, al obtener un archivo .HDF5 lo tenemos en la misma ruta que el .CSV. (También genera otro archivo pero no le daremos importancia en el presente manual).




Nombre	Fecha de modificación	Tipo	Tamaño
 yellow_tripdata_2018-01	05/08/2021 14:55	Archivo de valores...	754.003 KB
 yellow_tripdata_2018-01.csv.hdf5	21/08/2021 17:12	Archivo HDF5	1.394.427 KB
 yellow_tripdata_2018-01.csv.yaml	21/08/2021 17:12	Archivo YAML	1 KB

Figura 11.3: VAEX para leer el DataSet (parte 3)

Hemos leído el .CSV con VAEX y de paso, hemos obtenido el .HDF5.

Ahora, con ese .HDF5, podremos ver el motivo de trabajar con este formato.

Antes de nada explicaremos que es .HDF5, para entender un poco lo que estamos empleando.

12. HDF5 Y APACHE PARQUET: BREVE INTRODUCCIÓN

Características básicas de HDF5:

- | HDF5 se refiere a “Hierarchical Data Format Version 5”
- | HDF5 es open source
- | HDF5 puede almacenar muchos diferentes tipos de archivos en un mismo archivo.
- | HDF ha sido usado por la NASA durante muchos años
- | HDF5 es una tecnología que hace viable la gestión de datos muy grandes y complejos.

Existen muchas más cosas que se podrían aprender sobre HDF5, en nuestro caso lo que haremos será directamente tratar de usarlo con Python y Jupyter.

Respecto a las características básicas de Apache Parquet no hablaremos mucho dado que no será empleado en los ejemplos.

Pero ya se mencionó algo previamente, y además de ser Open Source Parquet también está relacionado con el ecosistema Hadoop.

Aunque, como decimos no entraremos en detalles, pero se basa en un tipo de almacenamiento “columnar storage” que difiere del .CSV que es “row-based”. (No tiene importancia, puedes quedarte con que es algo diferente que .CSV y en este tema no entraremos en más detalles.

Para DASK trataremos de leer directamente .CSV porque sus tiempos de lectura son muy rápidos, tal y como demostraremos en el presente manual, y cuando queramos leer archivos de muchas más filas, tal vez VAEX sería mejor opción, aunque habría que probar, obviamente.

13. ABRIMOS ESE DATASET CON VAEX (LEYENDO .HDF5)

Ahora que tenemos el HDF5, obtenido previamente, podemos abrirlo con VAEX, y ver el tiempo de lectura. Es tan simple como veremos a continuación.

VAEX desde HDF5

```
In [1]: %%time
# https://pypi.org/project/vaex/
# pip install vaex
import vaex

Wall time: 7.22 s
```

Figura 13.1: VAEX para leer el DataSet (parte 1)

```
In [2]: %%time
vaex_hdf5 = vaex.open("C:/Users/Manut/Desktop/apuntes_big_data_2/TEMA_6/data/yellow_tripdata_2018-01.csv.hdf5")
vaex_hdf5

Wall time: 5.11 s

Out[2]:
```

#	VendorID	tped_pickup_datetime	tped_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID
0	1	'2018-01-01 00:21:05'	'2018-01-01 00:24:23'	1	0.5	1	'N'	
1	1	'2018-01-01 00:44:55'	'2018-01-01 01:03:05'	1	2.7	1	'N'	
2	1	'2018-01-01 00:08:26'	'2018-01-01 00:14:21'	2	0.8	1	'N'	
3	1	'2018-01-01 00:20:22'	'2018-01-01 00:52:51'	1	10.2	1	'N'	
4	1	'2018-01-01 00:09:18'	'2018-01-01 00:27:06'	2	2.5	1	'N'	
...
8,759,869	1	'2018-01-31 23:21:35'	'2018-01-31 23:34:20'	2	2.8	1	'N'	
8,759,870	1	'2018-01-31 23:35:51'	'2018-01-31 23:38:57'	1	0.6	1	'N'	
8,759,871	2	'2018-01-31 23:28:00'	'2018-01-31 23:37:09'	1	2.95	1	'N'	
8,759,872	2	'2018-01-31 23:24:40'	'2018-01-31 23:25:28'	1	0.0	1	'N'	
8,759,873	2	'2018-01-31 23:28:16'	'2018-01-31 23:28:38'	1	0.0	1	'N'	

Tiempos:

- Con 1TB HDD sobre Windows: 5.11 segundos (alguna ocasión fueron 57 milisegundos)
- Con 1TB SSD sobre Linux: 40.1 milisegundos

Figura 13.2: VAEX para leer el DataSet (parte 2)

40 milisegundos para leer casi 9 millones de filas. No está nada mal !


```
In [3]: %%time
vaex_hdf5.describe()
```

```
INFO:MainThread:numexpr.utils:NumExpr defaulting to 4 threads.
```

```
Wall time: 9.56 s
```

Out[3]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count
data_type	int64	string	string	int64
count	8759874	8759874	8759874	8759874
NA	0	0	0	0
mean	1.560940488413418	--	--	1.6068553040831408
std	0.496272	--	--	1.258464
min	1	--	--	0
max	2	--	--	9

Tiempos:

- Con 1TB HDD sobre Windows: 9.56 segundos (a veces bastante más)
- Con 1TB SSD sobre Linux: 4.56 segundos

Figura 13.3: VAEX para leer el DataSet (parte 3)



Importante

En el caso de celdas donde hay estadística es normal que tarde un poco ya que son muchos cálculos, aun así, no está nada mal.

```
In [4]: %%time
        vaex_hdf5.fare_amount.value_counts()

INFO:MainThread:numexpr.utils:NumExpr defaulting to 4 threads.

Wall time: 1.62 s

Out[4]: 6.00      473234
        5.50      465176
        6.50      461927
        7.00      446376
        5.00      433249
        ...
        70.27         1
        70.01         1
        69.97         1
        69.87         1
        -450.00        1
Length: 1714, dtype: int64

Tiempos:


- Con 1TB HDD sobre Windows: 1.62 segundos (No ejecuta correctamente alguna vez)
- Con 1TB SSD sobre Linux: 114 milisegundos

```

Figura 13.4: VAEX para leer el DataSet (parte 4)

Ya podemos ver que VAEX es impresionante.

Pudiéramos hacer muchas más cosas, pero por el momento es suficiente.

Existe otra forma de obtener el archivo .HDF5 a través del .CSV y es lo que haremos a continuación.

No es relevante, pero no queremos que te despiste si en algún sitio lo ves escrito de esta forma.

14. VAEX: OTRA FORMA DE RESOLVERLO

VAEX (2ª forma)

```
In [1]: %%time
# https://pypi.org/project/vaex/
# pip install vaex
import vaex
```

Wall time: 7.28 s

Tiempos:

- Con 1TB HDD sobre Windows: 7.28 segundos
- Con 1TB SSD sobre Linux: 19.6 microsegundos

Figura 14.1: VAEX (otra forma) para leer el DataSet (parte 1)

```
In [2]: %%time
df_vaex_3 = vaex.from_csv("C:/Users/Manut/Desktop/apuntes_big_data_2/TEMA_6/data/yellow_tripdata_2018-01.csv")
df_vaex_3
```

INFO:MainThread:numexpr.utils:NumExpr defaulting to 4 threads.

Wall time: 1min 23s

```
Out[2]:
```

#	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID
0	1	'2018-01-01 00:21:05'	'2018-01-01 00:24:23'	1	0.5	1	'N'	41
1	1	'2018-01-01 00:44:55'	'2018-01-01 01:03:05'	1	2.7	1	'N'	239
2	1	'2018-01-01 00:08:26'	'2018-01-01 00:14:21'	2	0.8	1	'N'	262
3	1	'2018-01-01 00:20:22'	'2018-01-01 00:52:51'	1	10.2	1	'N'	140
4	1	'2018-01-01 00:09:18'	'2018-01-01 00:27:06'	2	2.5	1	'N'	246
...
8,759,869	1	'2018-01-31 23:21:35'	'2018-01-31 23:34:20'	2	2.8	1	'N'	158
8,759,870	1	'2018-01-31 23:35:51'	'2018-01-31 23:38:57'	1	0.6	1	'N'	163
8,759,871	2	'2018-01-31 23:28:00'	'2018-01-31 23:37:09'	1	2.95	1	'N'	74
8,759,872	2	'2018-01-31 23:24:40'	'2018-01-31 23:25:28'	1	0.0	1	'N'	7
8,759,873	2	'2018-01-31 23:28:16'	'2018-01-31 23:28:38'	1	0.0	1	'N'	7

Tiempos:

- Con 1TB HDD sobre Windows: 1 minuto 23 segundos
- Con 1TB SSD sobre Linux: 54.6 segundos

Figura 14.2: VAEX (otra forma) para leer el DataSet (parte 2)

```
In [3]: %%time
df_vaex_3.export('C:/Users/Manut/Desktop/apuntes_big_data_2/TEMA_6/data/yellow_tripdata_2018-01_exportado.hdf5')

Wall time: 1min 19s

Tiempos:
• Con 1TB HDD sobre Windows: 1 minuto 19 segundos (53.8 segundos en otro intento)
• Con 1TB SSD sobre Linux: 9.73 segundos
```

Figura 14.3: VAEX (otra forma) para leer el DataSet (parte 3)

```
In [4]: %%time
df_vaex_4 = vaex.open('C:/Users/Manut/Desktop/apuntes_big_data_2/TEMA_6/data/yellow_tripdata_2018-01_exportado.hdf5')
df_vaex_4

Wall time: 9.28 s

Out[4]:
```

	#	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID
0	1	1	2018-01-01 00:21:05	2018-01-01 00:24:23	1	0.5	1	N	41	41
1	1	1	2018-01-01 00:44:55	2018-01-01 01:03:05	1	2.7	1	N	239	239
2	1	1	2018-01-01 00:08:26	2018-01-01 00:14:21	2	0.8	1	N	262	262
3	1	1	2018-01-01 00:20:22	2018-01-01 00:52:51	1	10.2	1	N	140	140
4	1	1	2018-01-01 00:09:18	2018-01-01 00:27:06	2	2.5	1	N	246	246
...
8,759,869	1	1	2018-01-31 23:21:35	2018-01-31 23:34:20	2	2.8	1	N	158	158
8,759,870	1	1	2018-01-31 23:35:51	2018-01-31 23:38:57	1	0.6	1	N	163	163
8,759,871	2	1	2018-01-31 23:28:00	2018-01-31 23:37:09	1	2.95	1	N	74	74
8,759,872	2	1	2018-01-31 23:24:40	2018-01-31 23:25:28	1	0.0	1	N	7	7
8,759,873	2	1	2018-01-31 23:28:16	2018-01-31 23:28:38	1	0.0	1	N	7	7

Tiempos:

- Con 1TB HDD sobre Windows: 9.28 segundos (5.83 segundos en otro intento, 84 milisegundos en otro)
- Con 1TB SSD sobre Linux: 61.1 milisegundos

Figura 14.4: VAEX (otra forma) para leer el DataSet (parte 4)

En el último paso, donde indicamos 9.28 segundos, obtuvimos unos 84 milisegundos en la última ejecución realizada. Es por ello que se ha observado que los tiempos de ejecución pueden variar notablemente.

Obviamente, y como era de esperar, me ha vuelto a generar archivos en el mismo directorio raíz.

Podemos verlo en la siguiente figura:



Nombre	Fecha de modificación	Tipo	Tamaño
 yellow_tripdata_2018-01	05/08/2021 14:55	Archivo de valores...	754.003 KB
 yellow_tripdata_2018-01_exportado.hdf5	23/08/2021 12:06	Archivo HDF5	1.394.427 KB

Figura 14.5: VAEX (otra forma) para leer el DataSet (parte 5)

En este caso me ha generado solamente el .HDF5. Pero, como decimos, no tiene importancia en este momento.

A partir de ahora, te queda pendiente la realización de más pruebas, y de aprender más cosas.

Pero es posible afirmar que VAEX y DASK son 2 grandes herramientas.

Recomendamos que hagas tus propias pruebas.

Tal vez en la actividad de este tema se te pida algo al respecto.

15. PUNTOS CLAVE

- | Existen alternativas a PANDAS muy eficaces cuando queremos trabajar con enormes sets de datos.
- | Podemos hacer BIG DATA con un ordenador personal, pero será necesario, o recomendable disponer de un equipo potente si queremos que se ejecuten las celdas muy rápidamente, sobre todo si el set de datos es muy muy grande.
- | VAEX permite leer .CSV pero idealmente se trabajará con .HDF5
- | DASK es una gran herramienta, y en algunos casos pudiera ser ideal.

